WHAT IS CLAIMED IS:

1. An extensible, object-oriented, portable programming language that permits centrally defined resource management, wherein an object expressed by the language can be simple or compound, and wherein a simple object comprises the following attributes:

an object name;

an object type;

a version; and

defined accessibility; and

wherein a compound object comprises the following attributes:

an object name;

a base object;

a field;

defined accessibility; and

a persistence property.

2. The programming language of claim 1, wherein said compound object further comprises attributes selected from the group consisting of:

a version,

a child object,

a parameter,

a namespace,

a C++ abstract base type,

a volatile property,

an external property,

an inferior property, and

a ccdoc operator.

3. The programming language of claim 1, wherein said simple object can be emulated as an enumeration object.

4. The programming language of claim 1, wherein said field comprises state information.

5. The programming language of claim 4, wherein said state information comprises the following attributes:

a field name;

a field type;

an initial default value;

accessibility;

a construct property;

a destruct property;

an override property;

an automatic set function;

an automatic get function; and

a ccdoc operator.

6. The programming language of claim 1 having a syntax described by the following Syntax BNF:

letter ::= "A" | "B" |...| "Z",

number::= "1" |"2"|...|"9",

atomic_symbol::=letter atom_part,

atom_part::= empty | letter atom_part | number atom_part,

empty ::="",

S_expression::= atomic_symbol | "("S_expression "."S_expression")" | list,

list::= "("S_expression # S_expression ")".

7. The programming language of claim 1, wherein said language can express a nested list that comprises the following atomic elements:

keywords;

names of auto-generated elements; and

literals.

8. The programming language of claim 7, wherein said literals of the nested list are selected from the group consisting of booleans, numbers, and strings.

9. The programming language of claim 7, wherein said auto-generated elements are selected from the group consisting of objects and field names.

10. The programming language of claim 1, wherein said language can express hierarchically structured packages selected from the group consisting of applications and libraries.

11. The programming language of claim 10, wherein said packages can be defined by: (library (name, nameoflibrary) ...).

12. The programming language of claim 10, wherein said packages can be defined by: (library, nameoflibrary ...).

13. The programming language of claim 10, wherein said packages can be defined by: (application (name, nameofapplication) ...).

14. The programming language of claim 10 wherein said packages can be defined by: (application, nameofapplication...).

15. The programming language of claim 1, wherein the objects are selected from the group consisting of C++ classes, namespaces, templates, and constant values.

16. The programming language of claim 15, wherein said constant values are selected from the group consisting of enums, class static variables, and namespace-scoped global.

17. A system for describing structure of programming languages, comprising:

(a) a high-level programming language;

(b) an extensible, object-oriented programming language for describing said high-level programming language; and

(c) a programming tool for converting said object-oriented programming language.

18. The system of claim 17, wherein copyright text, CCDoc directives, and compiler pragmas are automatically added to the system.

19. The system of claim 17, wherein input and verification parameters are specified in said extensible and object-oriented descriptive programming language.

20. The system of claim 17, wherein said programming tool is a compiler.

21. The system of claim 17, wherein said programming tool is a translator.

22. A method for describing computer programs by retaining meta-information about program elements, thereby allowing optimization and functionality on multiple hardware and software platforms, comprising the following steps:

(a) creating a first program using a high-level programming language;

(b) creating a second corresponding program using an extensible, object-oriented programming language to describe the high-level source code; and

(c) converting the second corresponding program into a form of the high-level programming language.

23. The method of claim 22, wherein the form is machine-executable.

24. The method of claim 22, wherein the form is high-level programming language.

25. The method of claim 22, wherein results of said step (a) and said step (b) are placed into one file, and further comprising the steps of:

(d) copying said second corresponding program from the file; and

(e) combining said second corresponding program with the form of the high-level source code.

26. The method of claim 25, wherein the file is a header file.

27. The method of claim 26, wherein the header file comprises the following sections:

Definitions;

User Preamble;

User Pre-object;

User Member;

User Postobject; and

User Postamble.